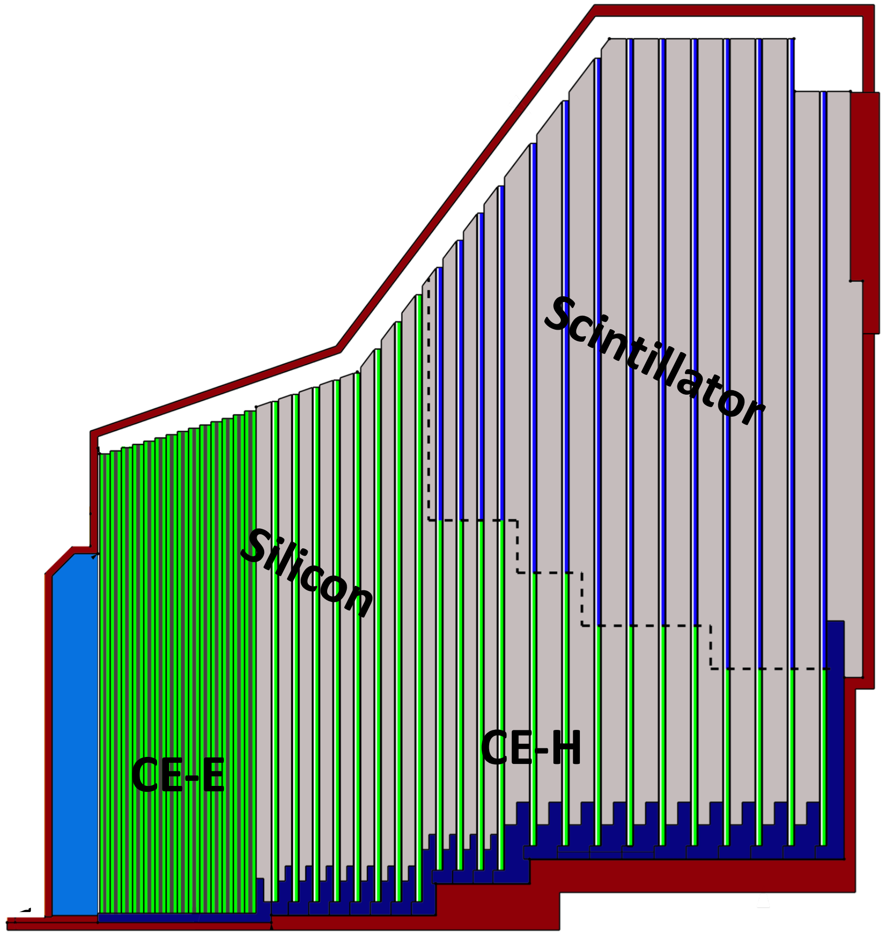
Design of a reconfigurable autoencoder neural network for detector front-end ASICs

CPAD 2021 – March 19, 2021

Columbia University : Giuseppe Di Guglielmo, Luca Carloni
Fermilab : Farah Fahim, Cristian Gingu, Christian Herwig, **Jim Hirschauer**,
Martin Kwok, Nhan Tran
Florida Tech : Danny Noonan
Northwestern University : Manuel Valentin, Yingyi Luo, Seda Memik

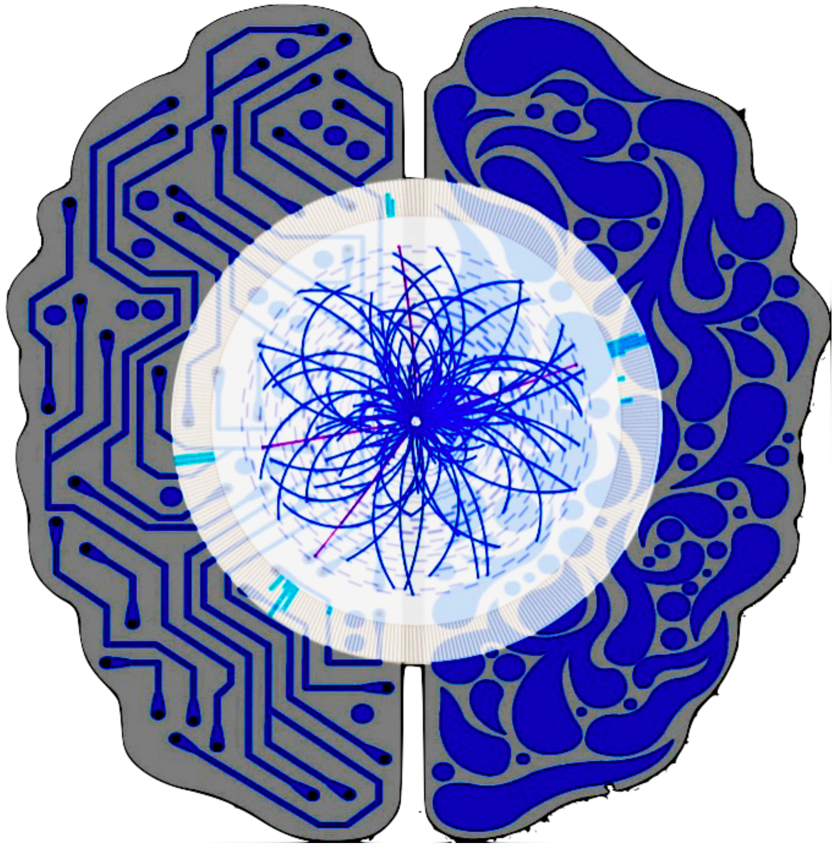
With thanks to the CMS Collaboration, and in particular, the CMS High–Granularity Calorimeter Group



Thanks also to

FAST MACHINE LEARNING LAB

https://fastmachinelearning.org/



2020 Fast ML for Science workshop:
https://indico.cern.ch/event/924283/

Please join the next workshop :
tentatively end-of-2021 / early-2022

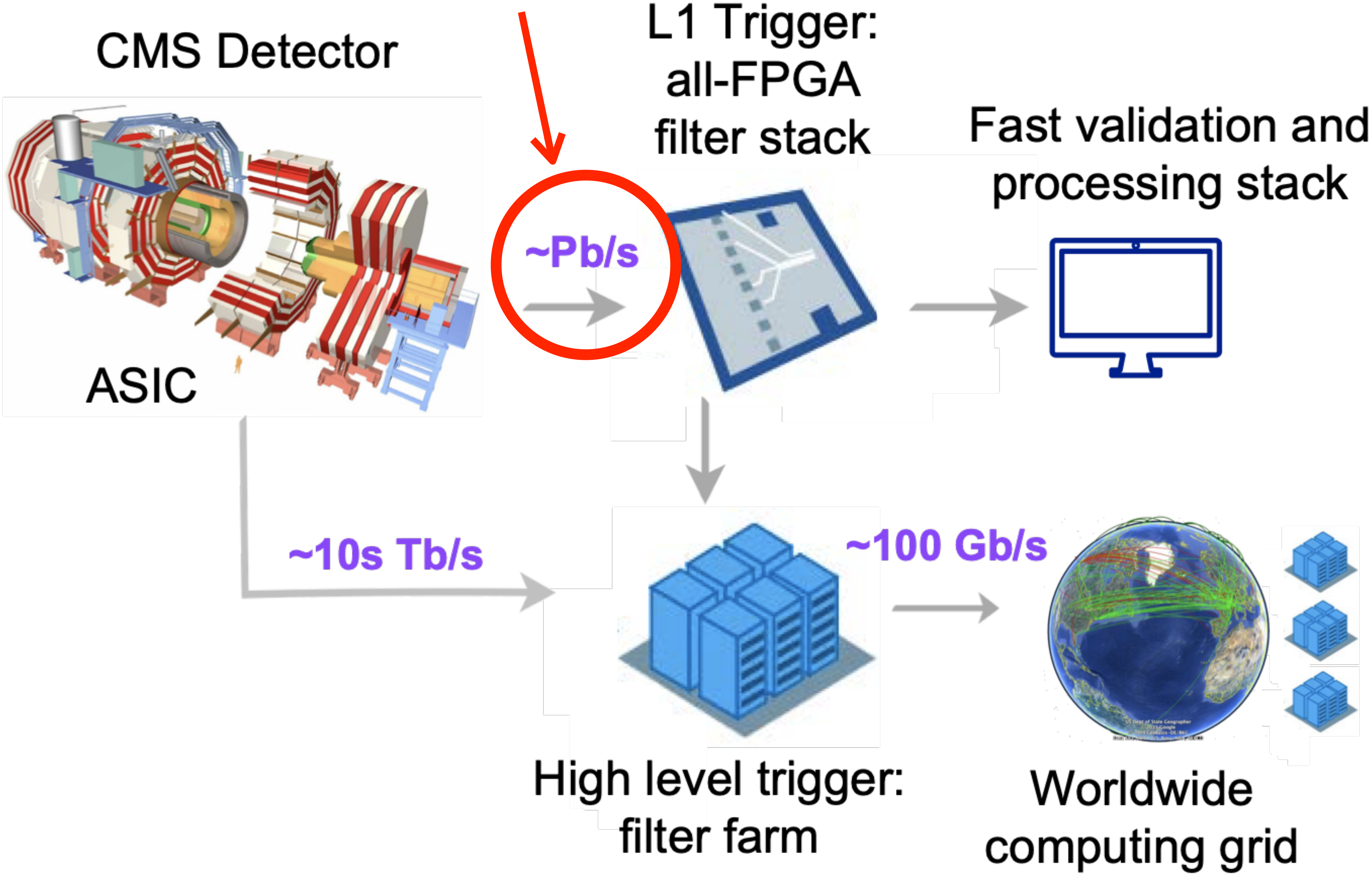
Motivation and introduction

- Higher **luminosity** ➔ higher occupancy ➔ higher **detector granularity** ➔ higher **data rates**
- Data challenge for **trigger path most severe** ➔ 40 MHz at HL-LHC
- Traditionally, on-detector electronics are kept as simple as possible.
- Data challenge ➔ **complex data processing must move to on-detector electronics**
  - object reconstruction (tracks, jets), object selection, **data compression**

- This talk: **Neural Network (NN) autoencoder** in **ASIC** for **on-detector data compression**.
- Design based on requirements for the CMS High-Granularity Calorimeter (HGCAL).
- Key features of design :
  - **low power, low latency, radiation tolerant** (200 Mrad, 1×10^7 20MeV-hadrons/cm^2/s)
  - **Fully re-configurable**:
    - **customize** the compression algorithm based on location within the detector
    - **adapt** the compression algorithm for changing detector and beam conditions

HL–LHC Data Challenge

Configurable on–detector data compression with machine learning



CMS Detector

ASIC

~Pb/s

L1 Trigger: all-FPGA filter stack

Fast validation and processing stack

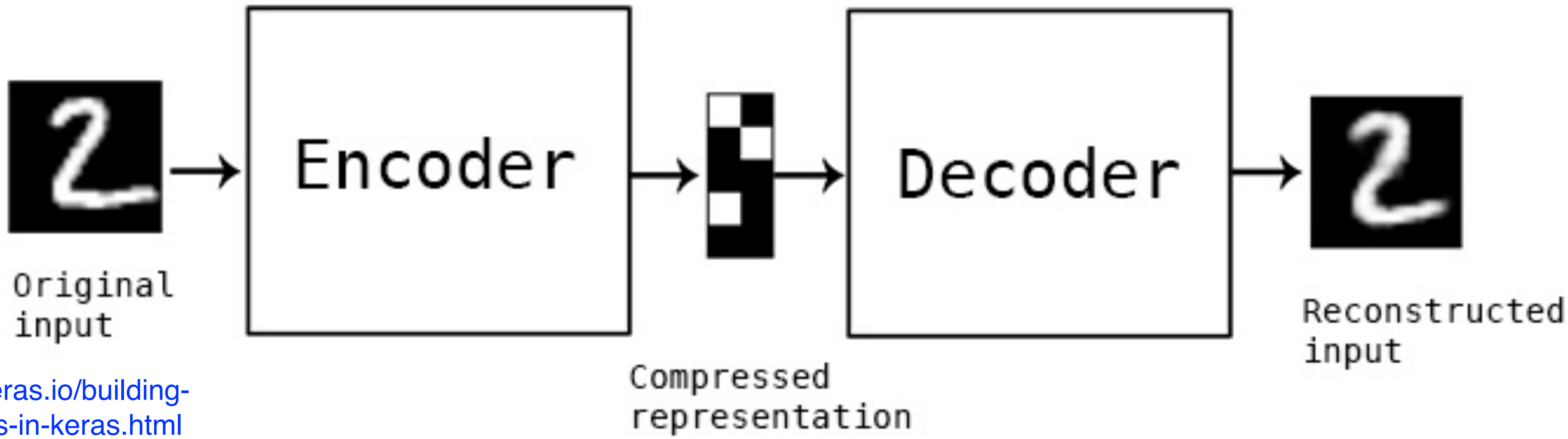
~10s Tb/s

~100 Gb/s

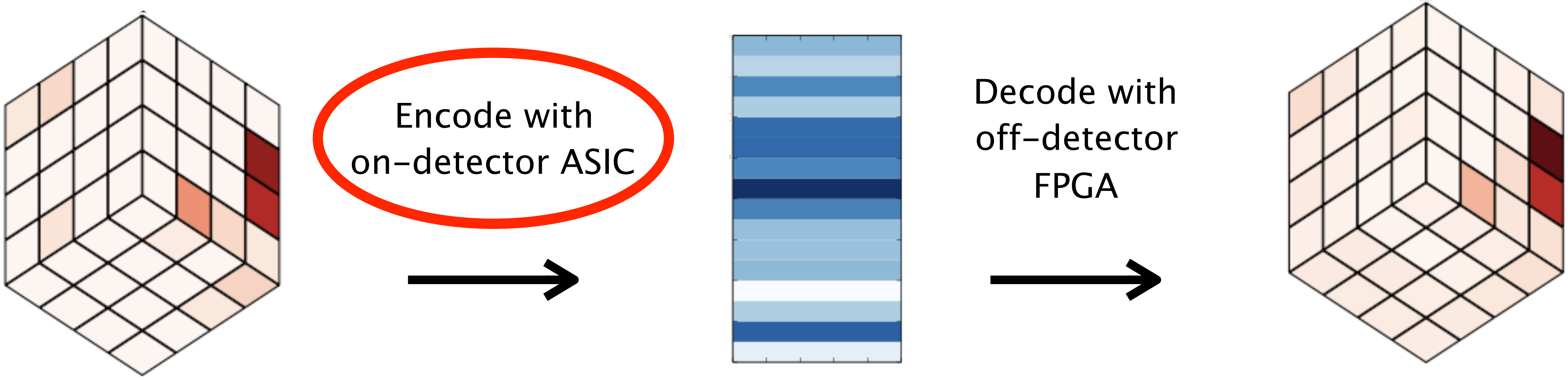
High level trigger: filter farm

Worldwide computing grid

Autoencoder concept



https://blog.keras.io/building-autoencoders-in-keras.html



Encode with on-detector ASIC

Decode with off-detector FPGA

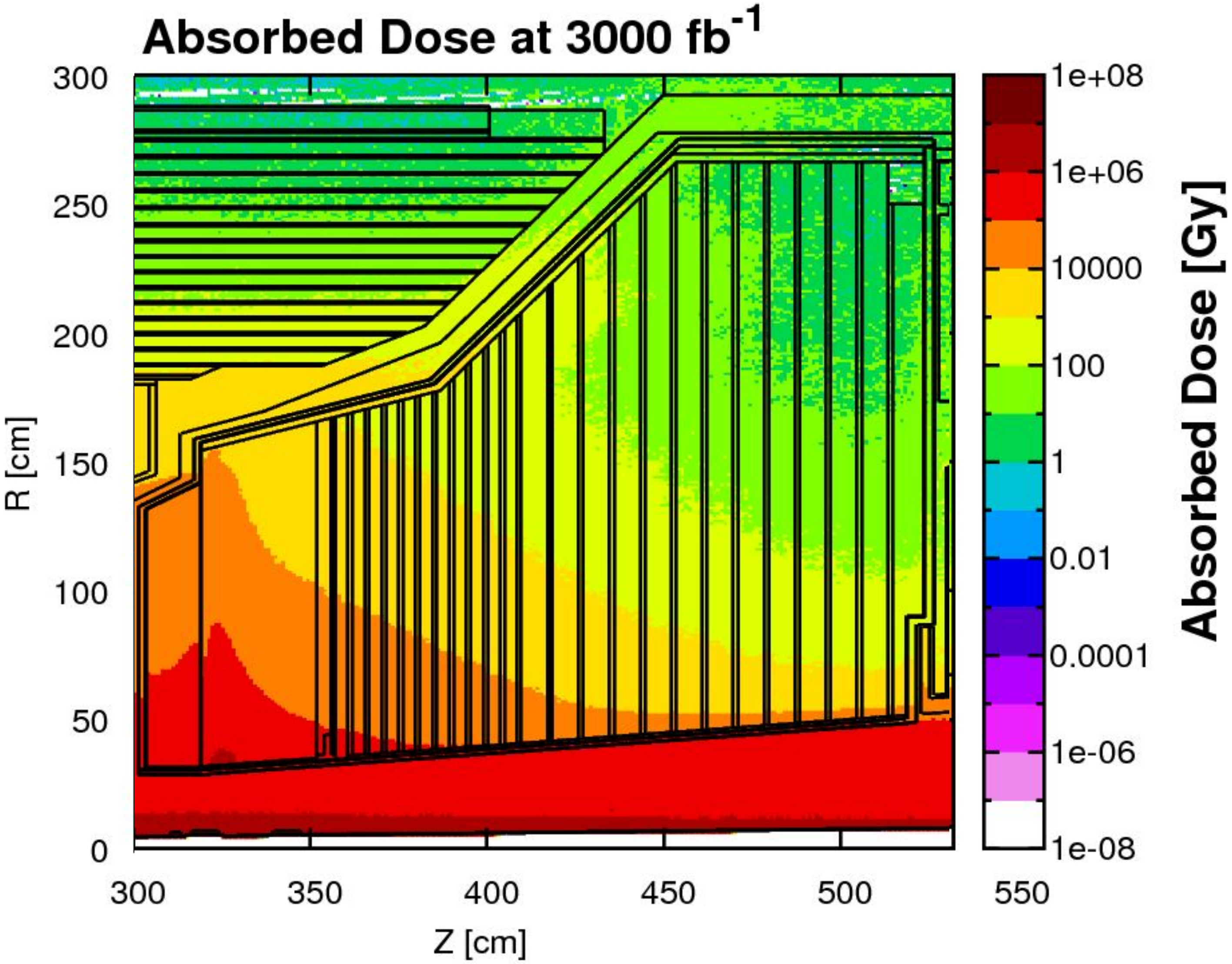
48-pixel input
336 bits

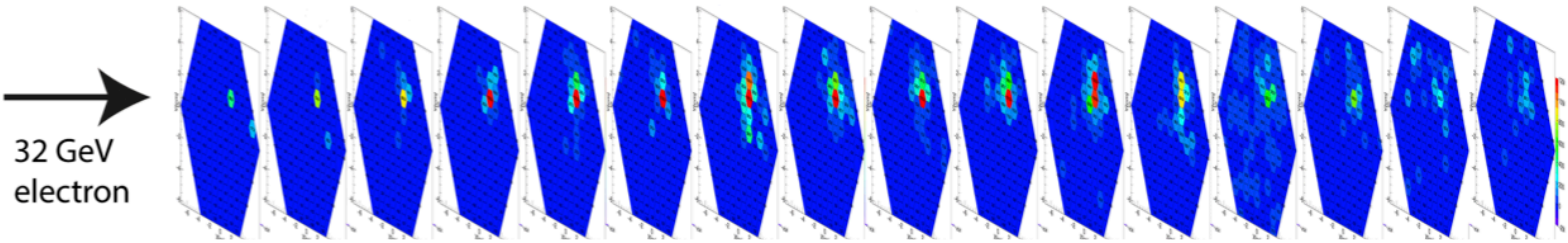
Transmit 16 × 3b outputs
48 bits

Decoded 48-pixel image

CMS High Granularity Calorimeter (HGCAL)

- "**Imaging calorimeter**" with ~6M readout channels.
- 50 layers of active material + absorber.
  - Front layers tiled with 300-500 8" hexagonal silicon modules.
- **HGCROC ASIC** : digitizes charge and arrival time and provides charge data for trigger path.
- **ECON ASIC** selects/compresses digital trigger data for transmission off-detector.
  - **NN Encoder to be included in ECON**.





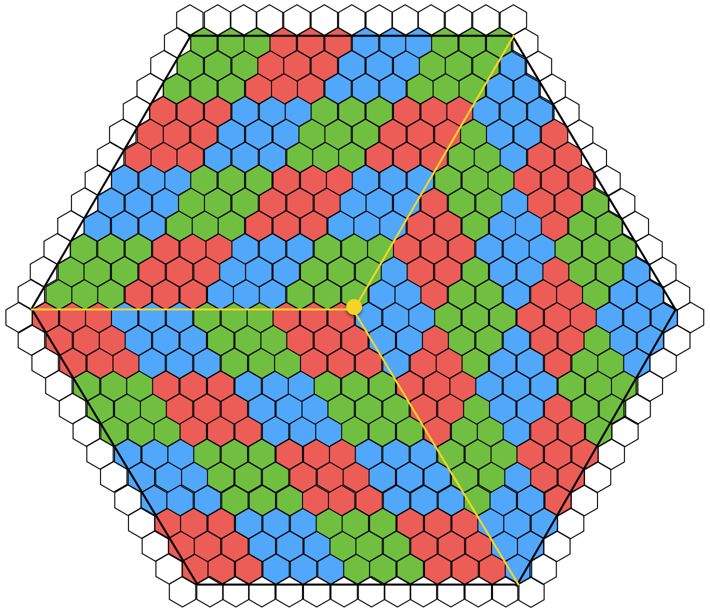
HGCAL trigger data challenge

Trigger path stage	Number channels	bits/ channel	Average Compression factor	Data rate*	# links* (10.24 Gbps)
Raw data	6M	20	1	5 Pb/s	1M
Hardware reduction	1M	7	1	300 Tb/s	60k
Threshold selection	1M	7	7	40 Tb/s	9k

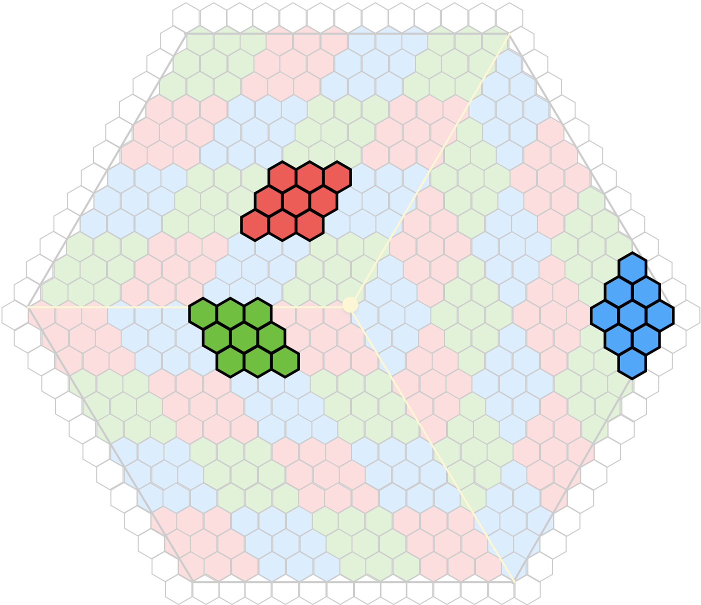
* Assumes 40 MHz rate and 50% link packing efficiency

- **Baseline HGCAL design for trigger selection in ECON** : threshold algorithm in ECON selects trigger cells with charge exceeding a threshold.

HGCAL 8" hex module



432 silicon sensors ➔ 48 trigger cells (TC) @ 7b per TC



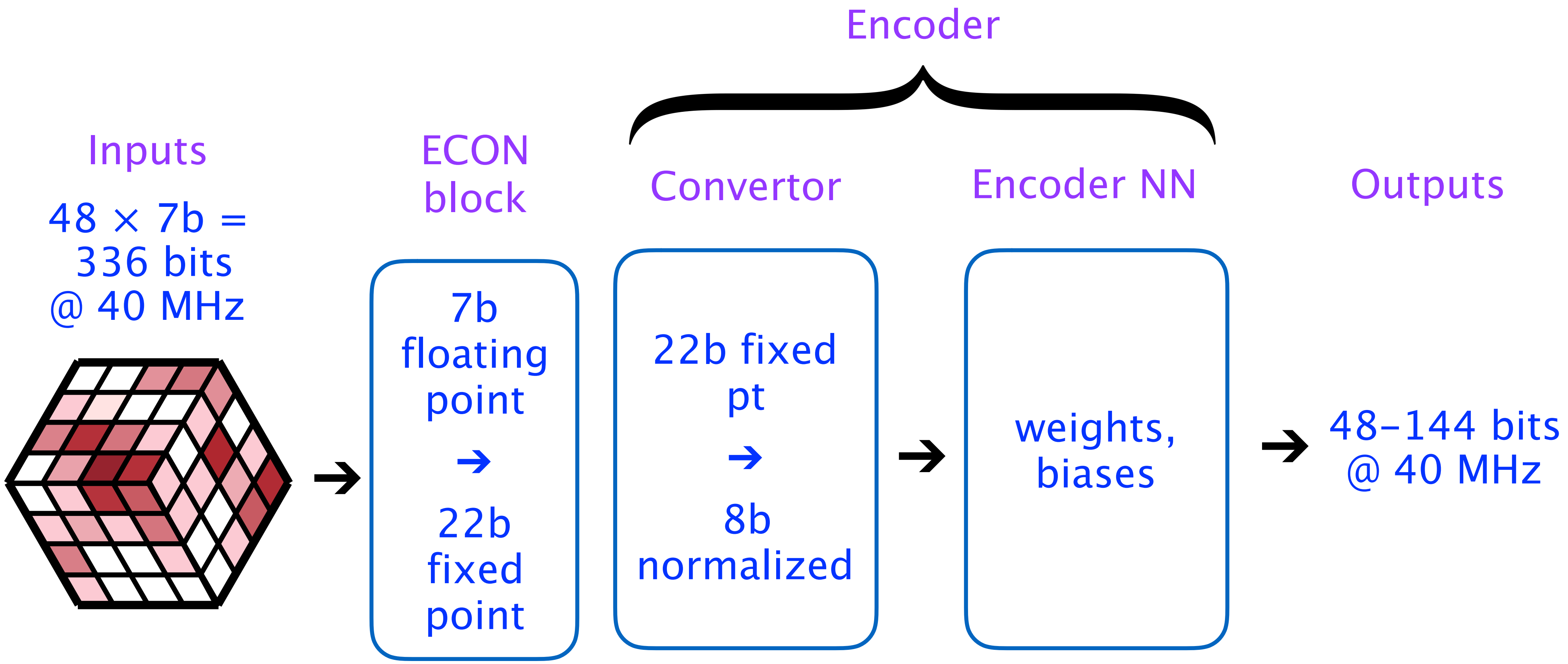
Traditional threshold algorithm : 3 of 48 TC readout for most of detector (2 × 1.28G elink per module)

Encoder design considerations

- **Minimize** : **power** (< 100 mW) + **area** (< 4 mm^2) + **latency** (< 100 ns)
- **Maximize** : **physics performance** + **configurability** + **radiation tolerance**

- **Network architecture** and precision of weights and biases: fixed in design

- **Fully re-configurable** : **all network weights and biases** + dimensionality of output



Encoder

Inputs

48 × 7b = 336 bits @ 40 MHz

→

ECON block

7b floating point → 22b fixed point

Convertor

22b fixed pt → 8b normalized

→

Encoder NN

weights, biases

→

Outputs

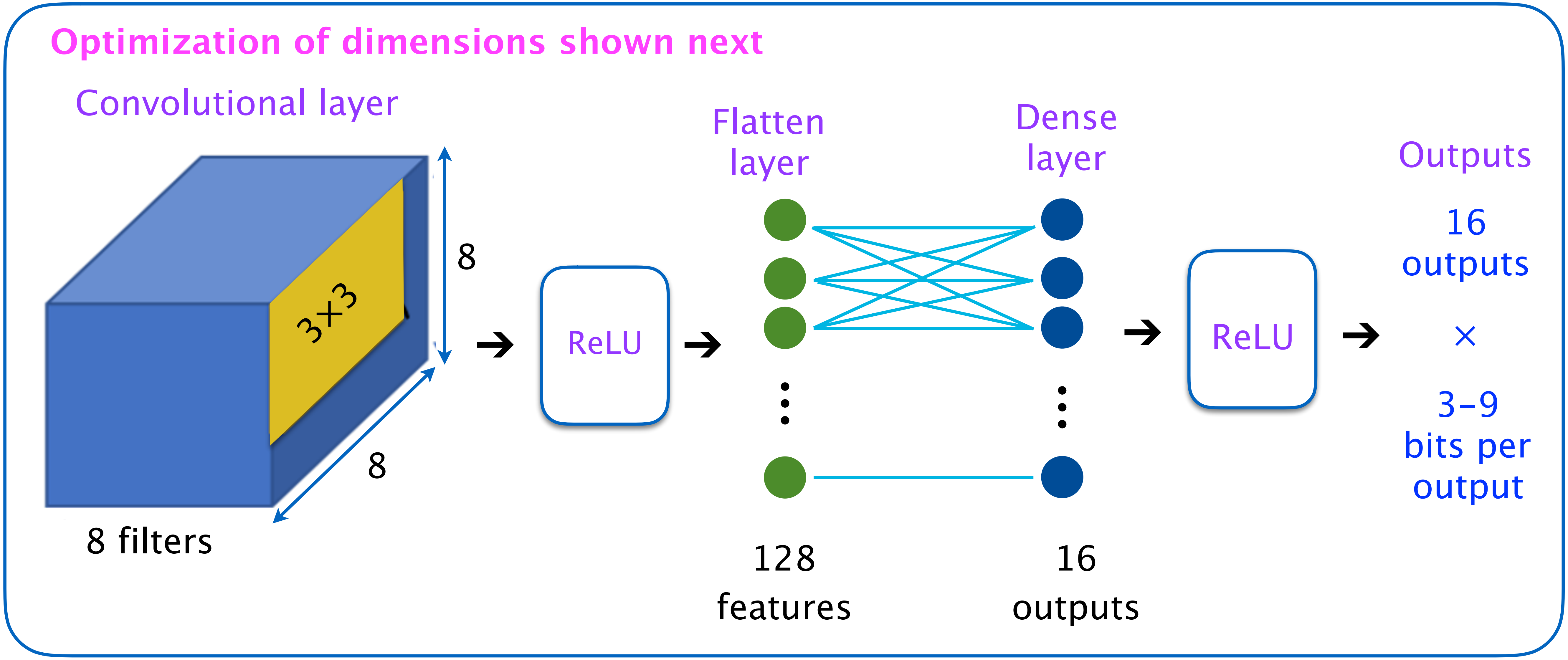
48–144 bits @ 40 MHz

Encoder NN design considerations

Encoder NN components

- **Convolutional layer** (conv2D): extract geometric features
- **Flatten layer** : vectorizes 2D image from conv2D (128 = 8 × 4 × 4)
- **Dense layer :** decide which geometric features are important
- **ReLU :** activation function

Encoder NN



Optimization of dimensions shown next

Convolutional layer

3×3

8

8

8 filters

→ ReLU →

Flatten layer

128 features

Dense layer

16 outputs

→ ReLU →

Outputs

16 outputs

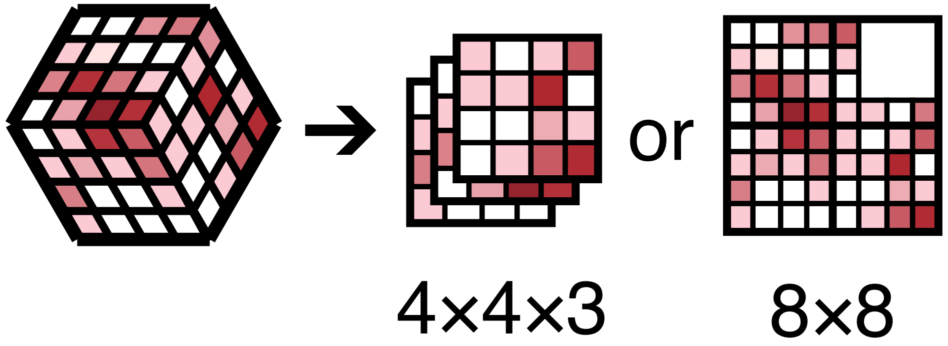
×

3–9 bits per output

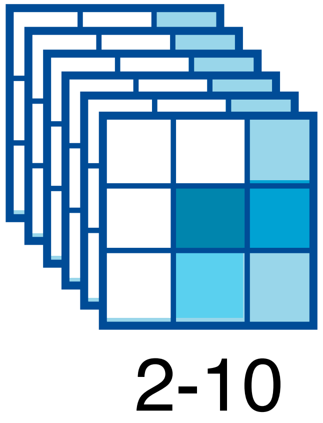
Encoder NN architecture optimization

- Optimize encoder network architecture choices including :

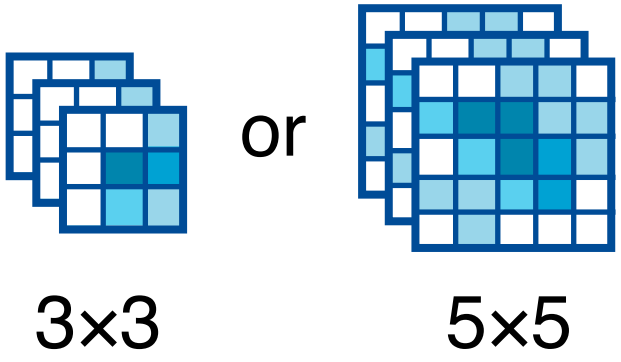
Geometry mapping



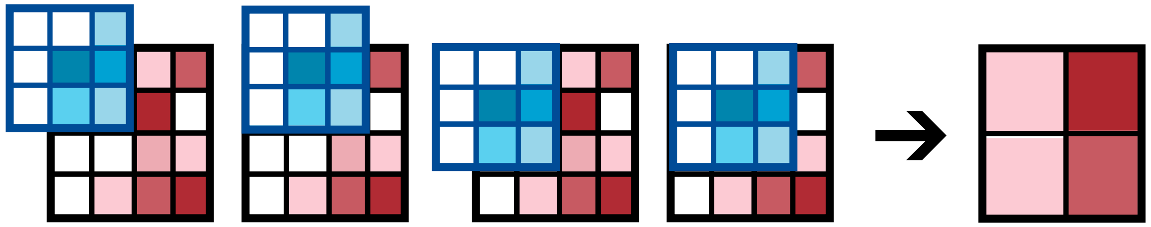
# of conv2D filters



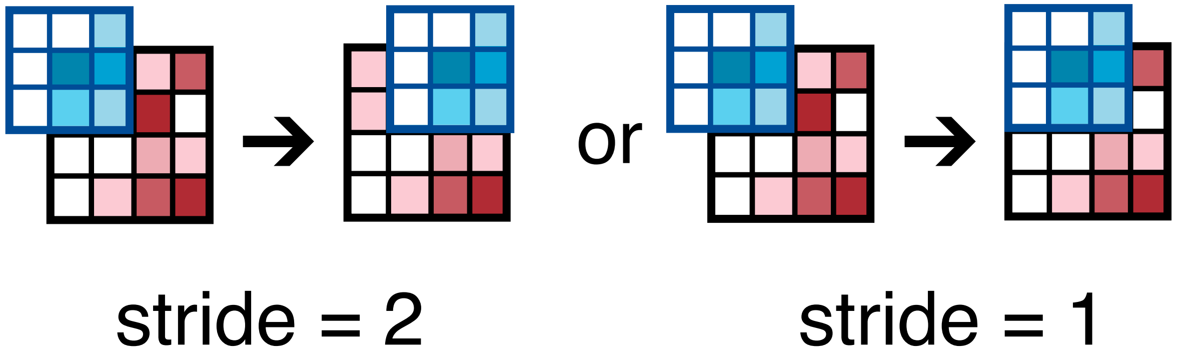
conv2D kernel size



Max pooling conv2D outputs

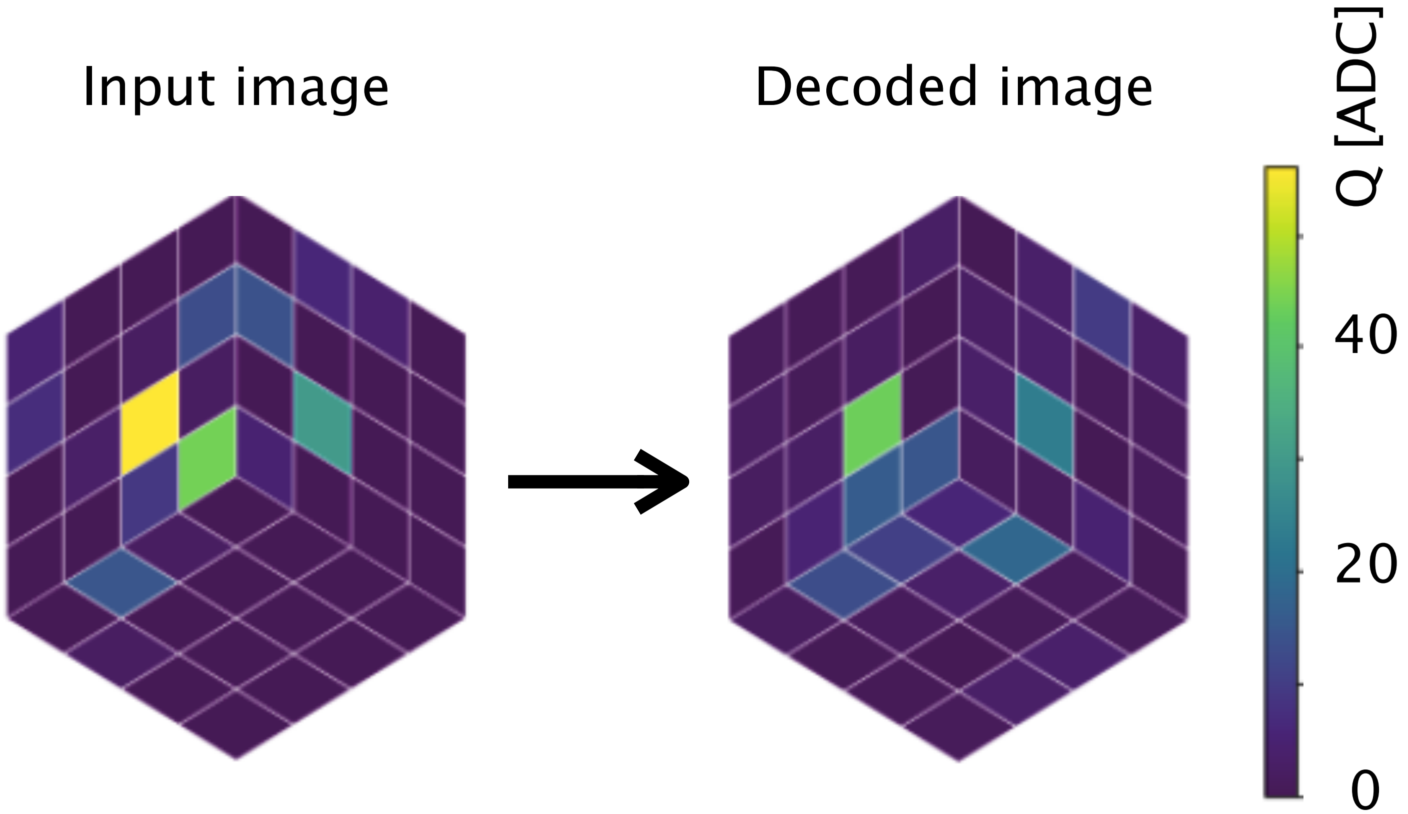


conv2D kernel stride

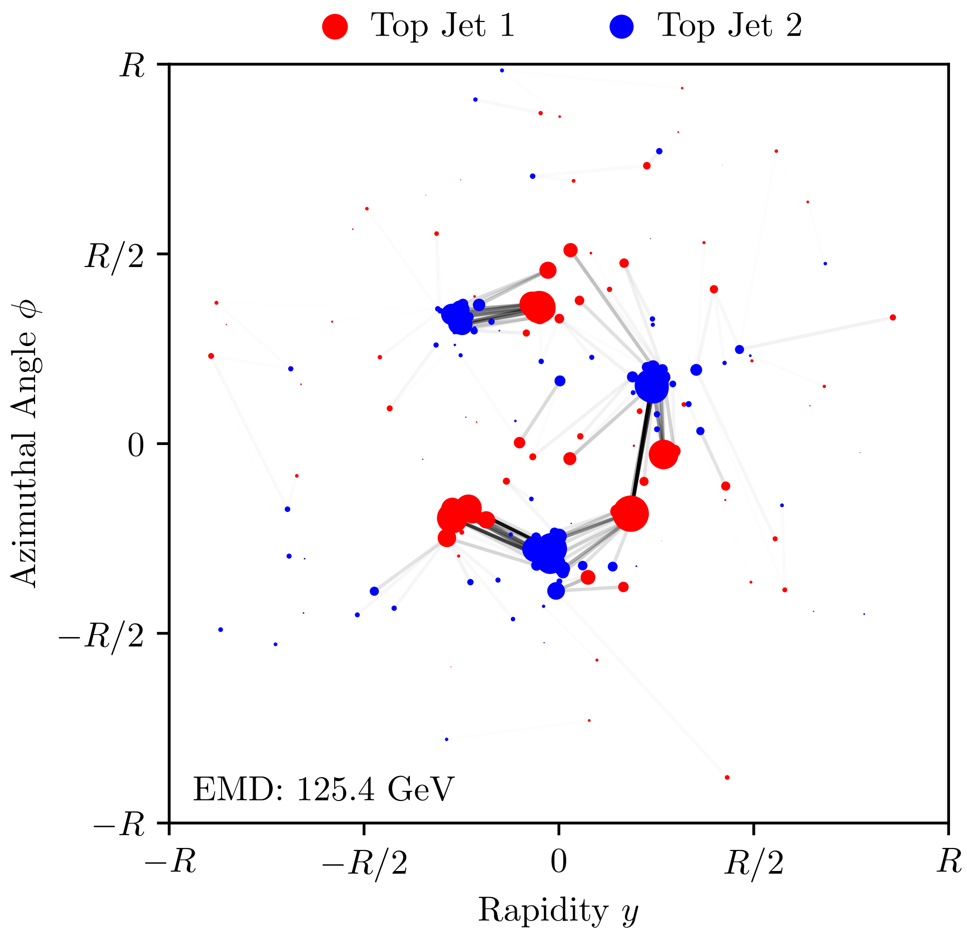


Performance metric : EMD

- Judge network performance according to **image similarity**.
- **Energy Mover's Distance** : quantify the cost of transforming one image into another as energy × distance.
- For each NN variation : train network and evaluate with simulated physics events including top quarks (jets, leptons) and **200 pileup**.



arXiv:1902.02346
Komiske, Metodiev, Thaler

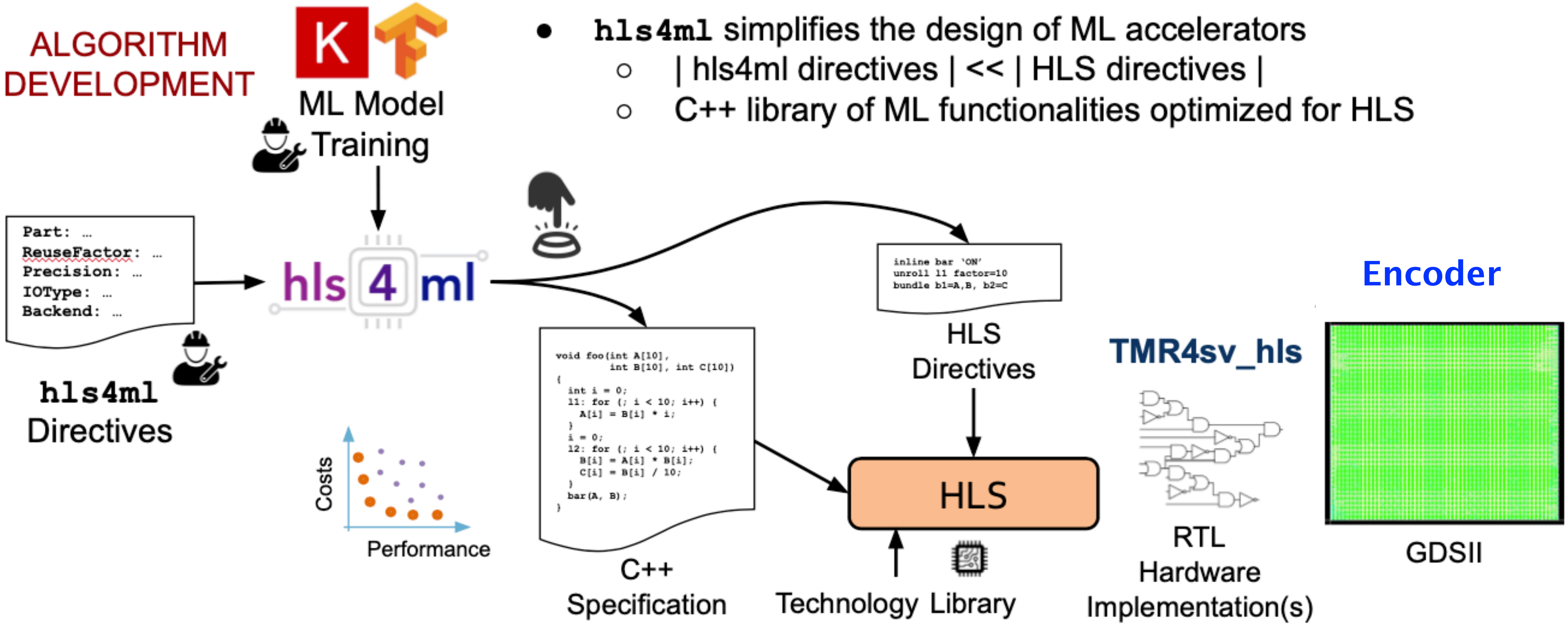


Physics driven hardware co-design

Rapid prototyping and optimization of network achieved through

- **QKeras** : network development with **quantization-aware training** and physics simulation
- **hls4ml** : neural network description (h5 file e.g.) ➔ HLS-compliant C++ format
- **Catapult HLS** : C++ ➔ RTL
- **TMR4sv_hls** : Automated TMR for System Verilog

- **hls4ml** simplifies the design of ML accelerators
    - | hls4ml directives | << | HLS directives |
    - C++ library of ML functionalities optimized for HLS



```
Part: …
ReuseFactor: …
Precision: …
IOType: …
Backend: …
```

```
void foo(int A[10],
         int B[10], int C[10])
{
  int i = 0;
  l1: for (; i < 10; i++) {
    A[i] = B[i] * i;
  }
  i = 0;
  l2: for (; i < 10; i++) {
    B[i] = A[i] * B[i];
    C[i] = B[i] / 10;
  }
  bar(A, B);
}
```

```
inline bar 'ON'
unroll l1 factor=10
bundle b1=A,B, b2=C
```

Rapid design optimization

- **Power and area** : roughly scale with number of model operations and parameters
- **Performance** : EMD mean and RMS are both important

Lower EMD is better

	Network Architecture					Relative Power & Area		Relative Performance	
Test feature	**Geometry**	**# filter**	**kernel**	**stride**	**pooling**	**# params**	**# operations**	**EMD Mean**	**EMD RMS**
Reference	4x4x3	8	3x3	1	none	1.00	1.00	1.00	1.00
4x4x3 -> 8x8	**8x8**	8	3x3	1	none	2.73	1.76*	0.64	0.41
max pooling	8x8	8	3x3	1	**2x2**	0.71	0.97*	0.59	0.33
3x3 -> 5x5 kernel	8x8	8	**5x5**	1	2x2	0.99	2.76	0.64	0.35
pooling -> stride=2	8x8	8	3x3	**2**	**none**	**0.94**	**0.59**	**0.76**	**0.46**
8 -> 10 filters	8x8	**10**	3x3	2	none	1.17	0.73	0.73	0.43
8 -> 6 filters	8x8	**6**	3x3	2	none	0.70	0.44	0.85	0.57

* zero operations removed

- **Reference design** : presented in Fall 2020**

- **Final design** : 8×8 geometry + 8 filters + 3×3 kernel + stride =2
  - **50% power** and 80% area of reference (from simulation)
  - **2× better performance** (EMD RMS) than reference

** https://indico.cern.ch/event/924283/contributions/4105329/attachments/2152250/3630590/encoder_asic_fastml2020.pdf
https://www.eventclass.org/contxt_ieee2020/online-program/session?s=N-34#e280
https://www.eventclass.org/contxt_ieee2020/online-program/session?s=N-24#e189

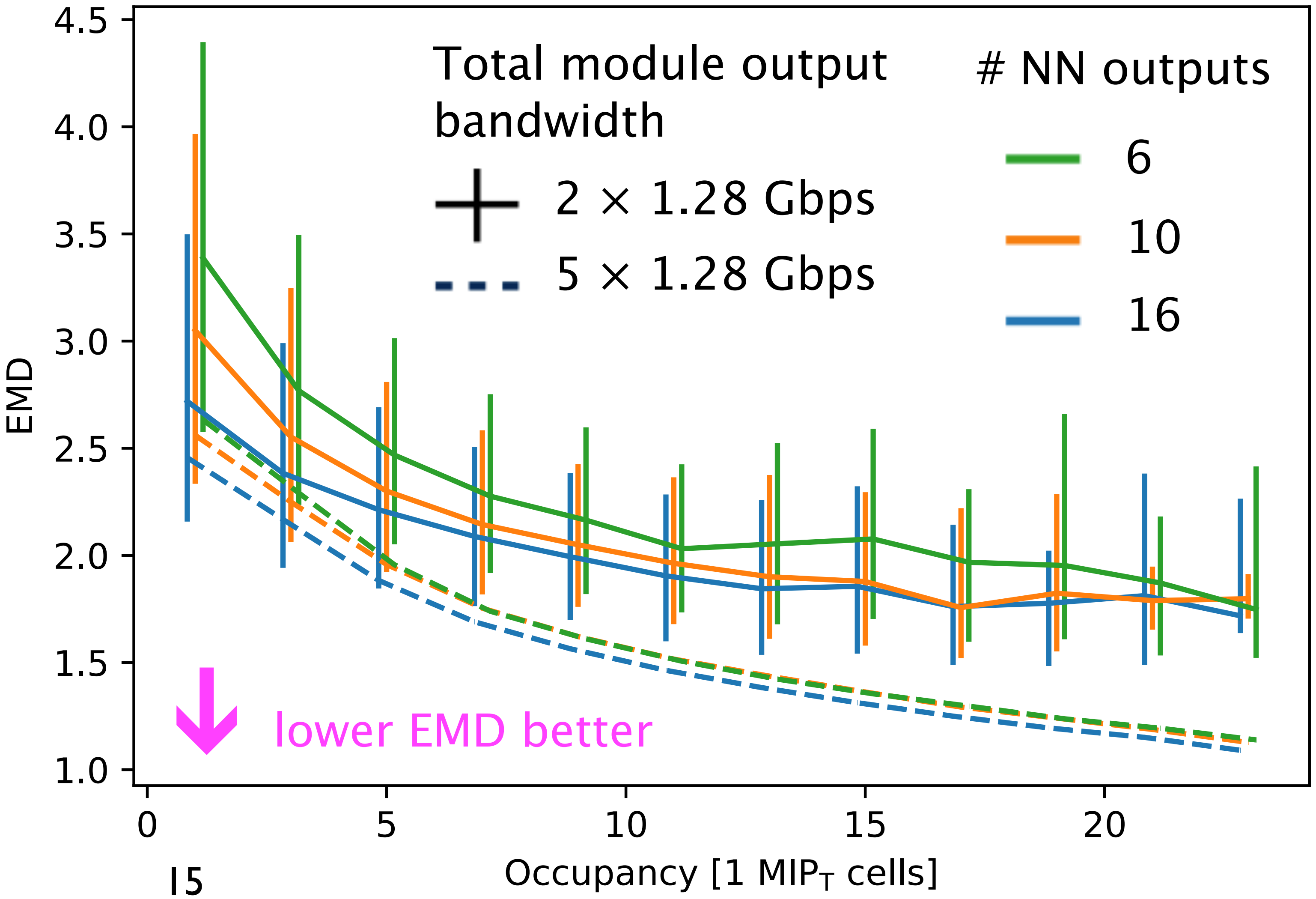
Optimization of NN output

- Better to use **many low-precision** or **fewer high-precision outputs**?
- Compare EMD performance keeping power and area fixed.
- Conclusion : **more lower-precision outputs is better**
  - for both high- and low-bandwidth scenarios
  - for full range of module occupancy

ECON ASIC allows user to **select any of 16×9 output bits** for transmission

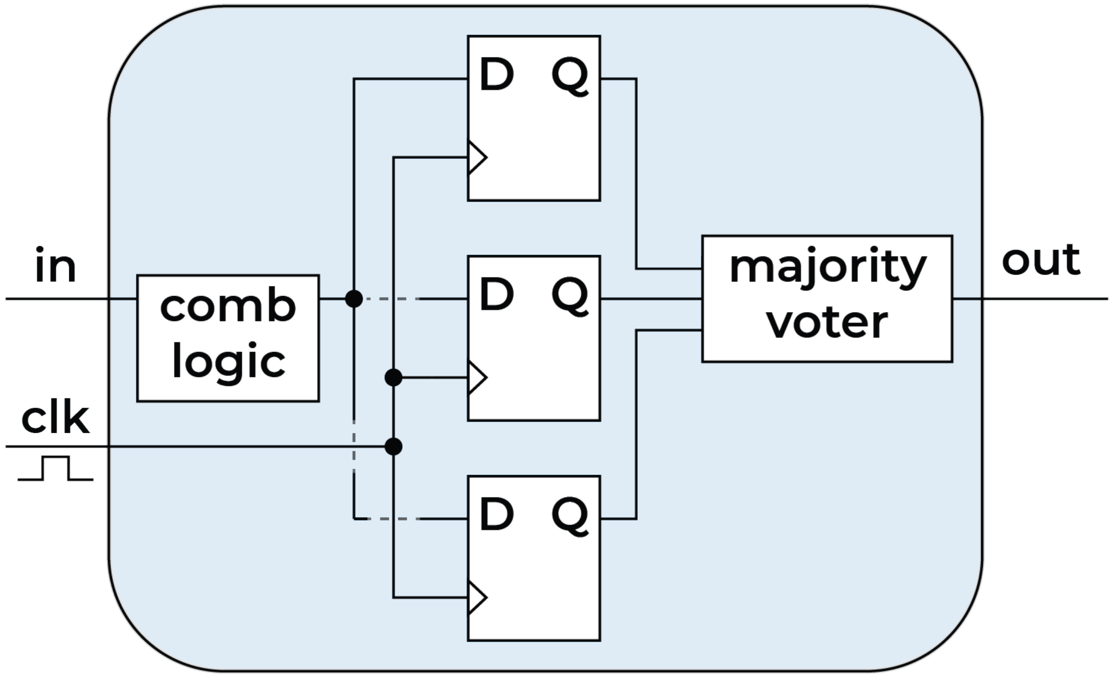
- Expect to use 16 × 3 (9) bits for low (high) occupancy zones.

- Corresponding precision used in **QKeras quantization-aware training optimizes network** for programmed output configuration.



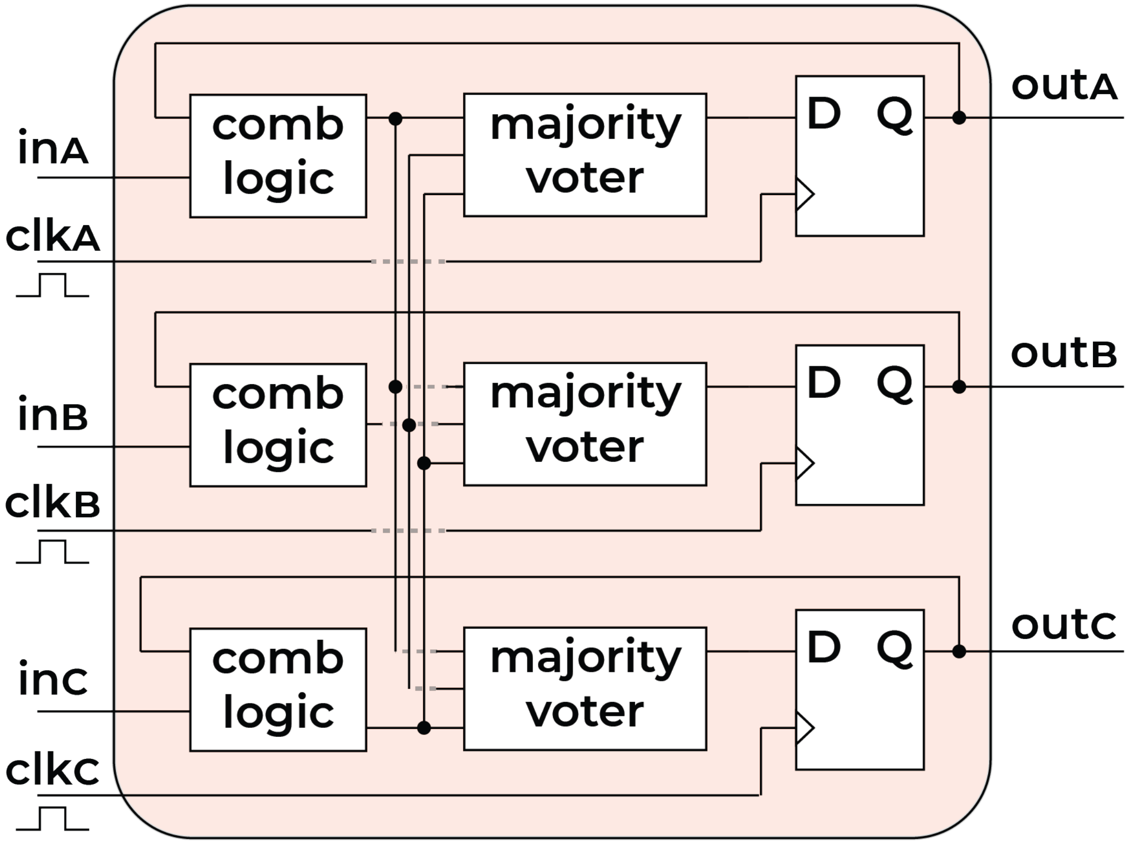
Single event effect mitigation

Data path :
Encoder & Convertor



- New data every 25ns
- Triplicate registers
- No auto-correction

Configuration : I²C secondary

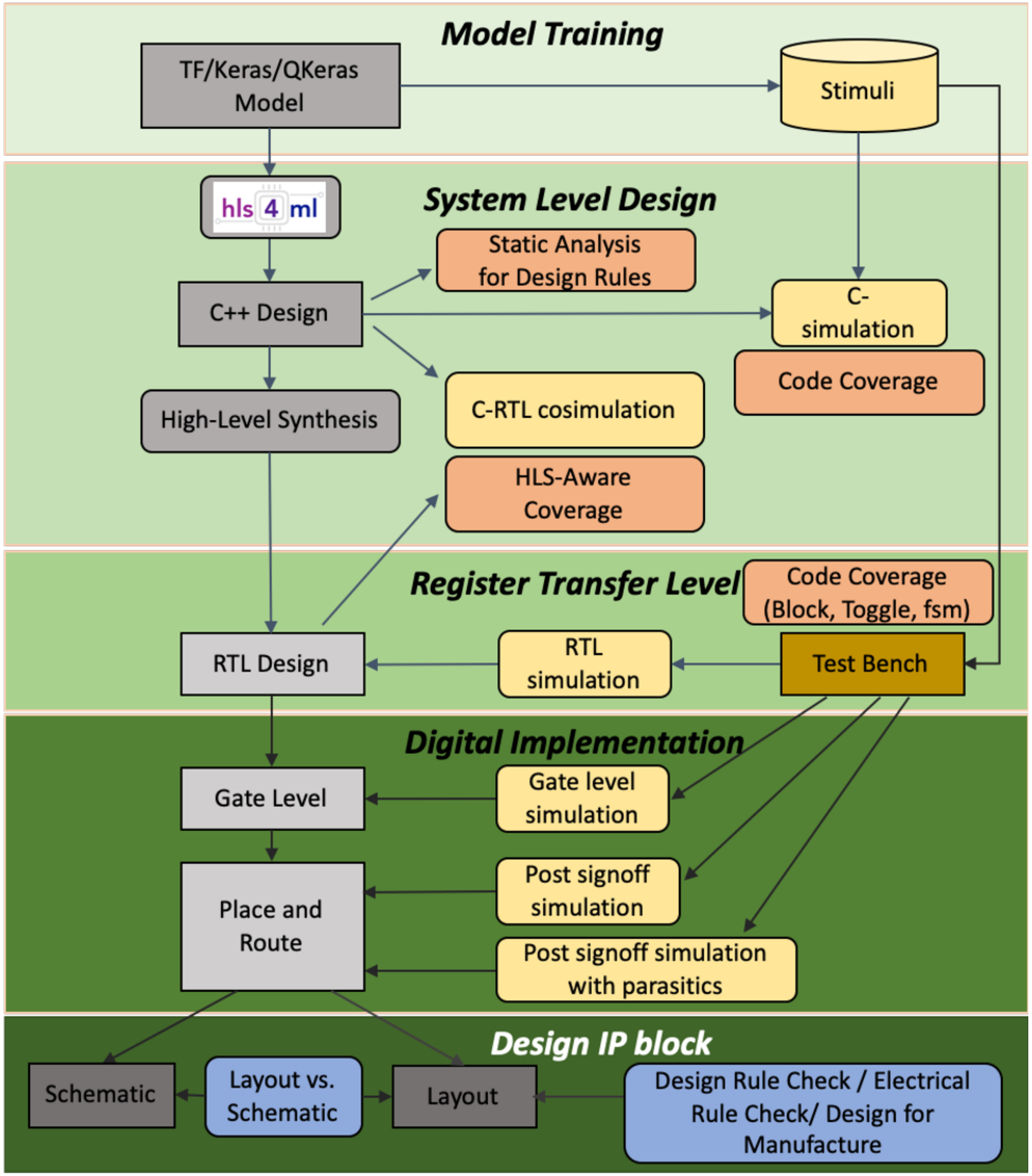


- Long term weights storage
- Triplicate registers, logic, and clocks
- Auto-correction included

Design and verification methodology

Verification performed at each stage of design:

- Model training
- hls4ml
- Catapult HLS
- RTL
- Synthesis
- Place and route
- LVS and DRC



Design and verification methodology

Step	Type	Run Time	Iterations	Size
Model generation	D	1s	50–100	1.1k lines of C++
C Simulation	V	1s		
HLS	D	30 min	3–100	40k lines of verilog
RTL simulation	V	1 min		
Logic synthesis	D	6 hrs	6	750k gates
Gate-level sim	V	30 min		
Place and route	D	50 hrs		780k gates
Post-layout sim	V	1 hrs		
Post-layout parasitic sim	V	2 hrs		
SEE simulation	V	4 hrs		
Layout	D	20 min	1	7.6M transistors
LVS and DRC	V	1 hr		

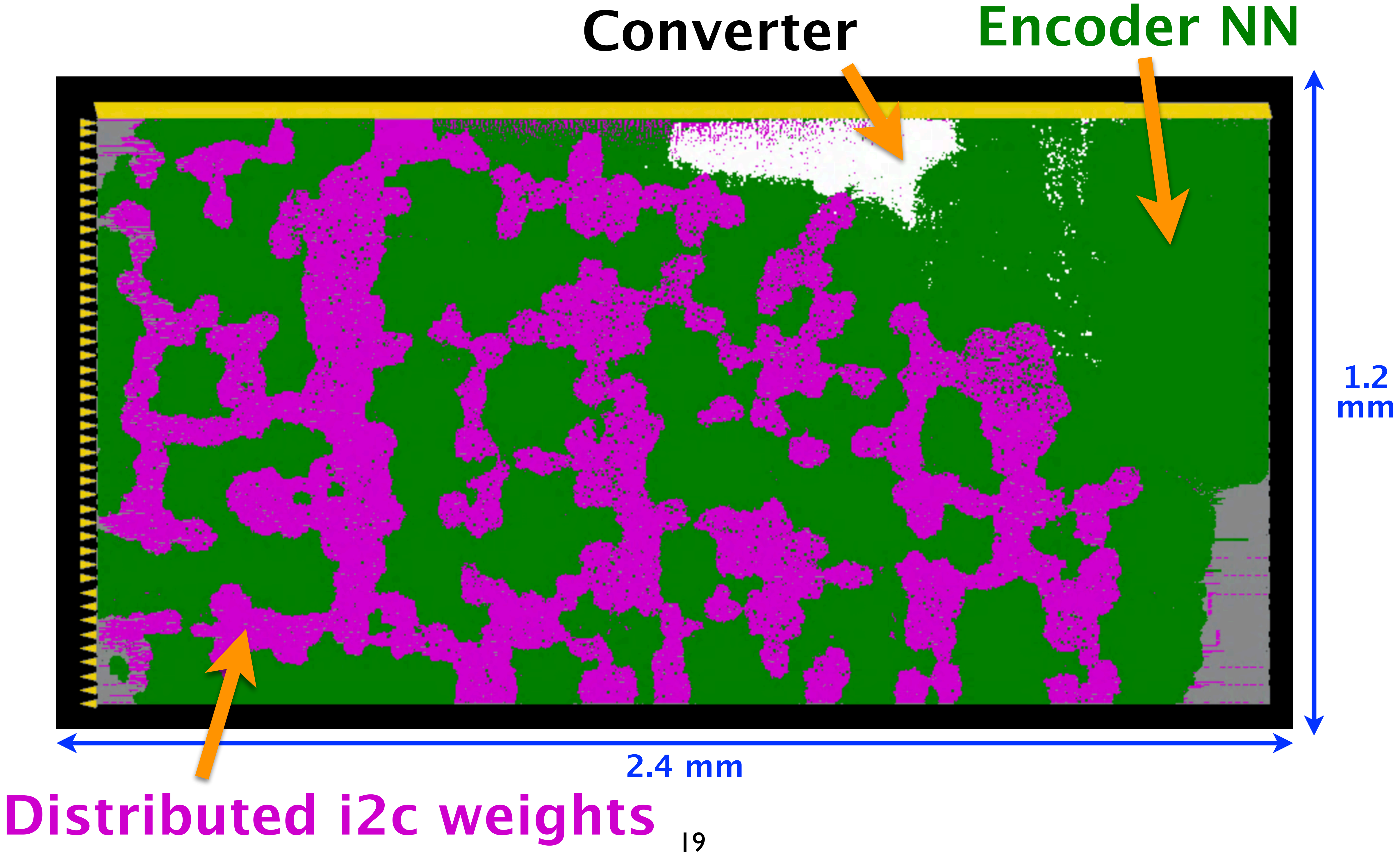
Network optimization

Design optimization

Increasing time and complexity

Place and route

- Integrated design to avoid routing congestion from 14k bits of weights (programmable via I²C) connected from periphery.



Design Performance Metrics

Physics performance studies in progress ➔ preliminary performance with non-optimized training **comparable to traditional threshold algorithm.**

Requirements	
Rate	40 MHz
Total ionizing dose	200 Mrad
High energy hadron flux	1×10^7 cm^2/s

Metric	Simulation	Target
Power	48 mW	<100 mW
Energy / inference	1.2 nJ	N/A
Area	2.88 mm^2	<4 mm^2
Gates	780k	N/A
Latency	50 ns	<100 ns

* EMD RMS

Summary

- **Autoencoder neural network for on-detector data compression**.
  - Low power, low latency, radiation tolerant, fully re-configurable
  - 65nm LP CMOS

- **Established design and verification methodology** based on **hls4ml + Catapult HLS** allows rapid progression from algorithm development through circuit implementation.

- Optimized network provides **2× better performance** at **~50% power** of reference network.

Acknowledgements

- **ECON design team for inclusion in ECON ASIC :** Davide Braga, Mike Hammer, Jim Hoff, Paul Rubinov, Alpana Shenai, Cristina Mantilla Suarez, Chinar Syal, Xiaoran Wang, Ralph Wickwire

- **CMS HGCAL for simulated training images**
  - Jean-Baptiste Sauvan for simulation development
  - Andre Davide for useful discussion on network optimization

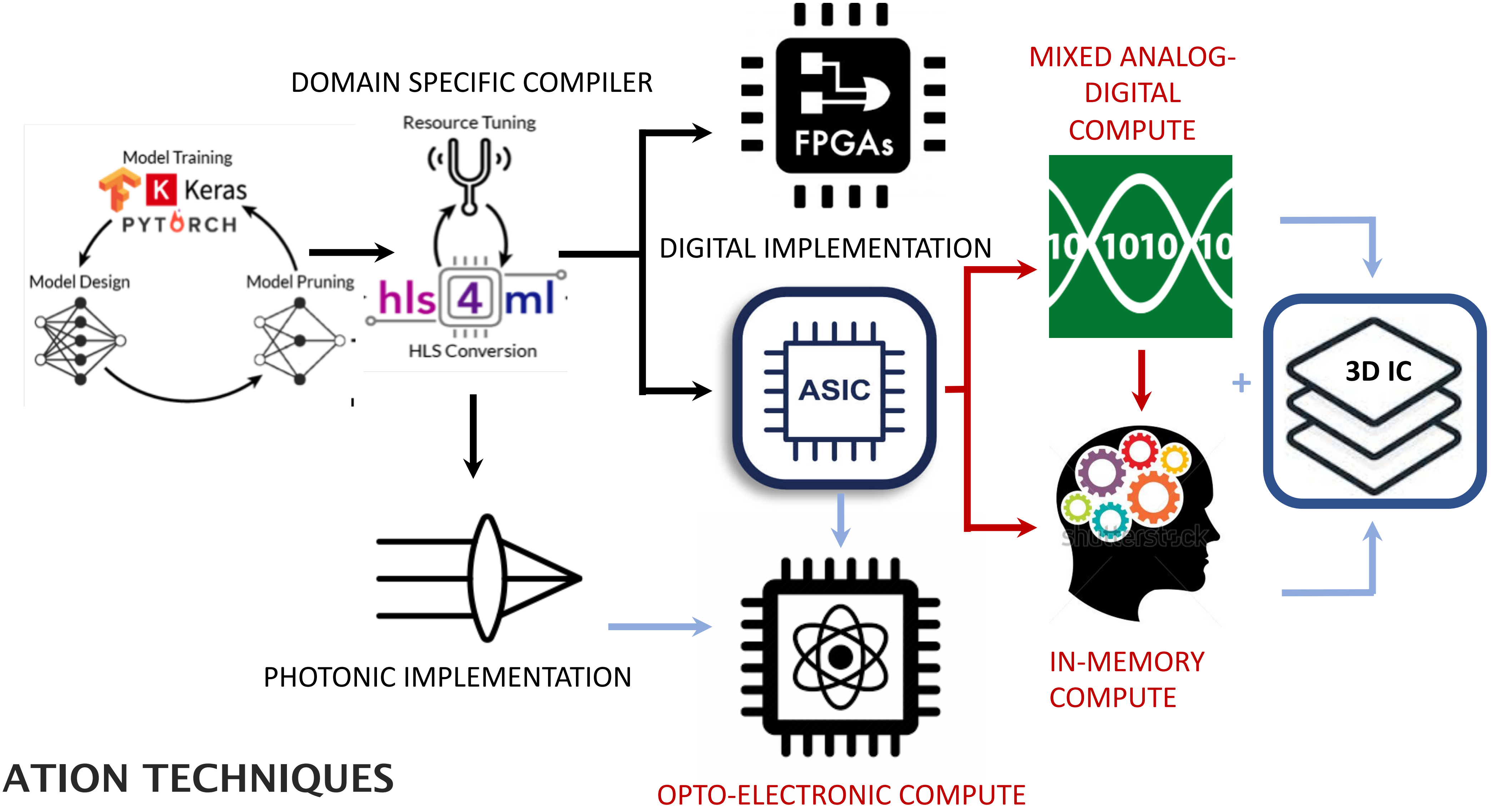
- **hls4ml developers** : Javier Duarte, Phil Harris, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Sioni Summers https://fastmachinelearning.org/hls4ml/

- **Mentor/Siemens Catapult HLS** : Sandeep Garg and Anoop Saha

- **Cadence Innovus and Incisive** : Bruce Cauble and Brent Carlson

Additional material

Future : towards heterogenous intelligent system on-chip



OPTIMIZATION TECHNIQUES

- Analog Mixed-Signal Kernels
- In-memory compute e.g. with using memristors (non-Von Neumann approaches)
- Neuromorphic computing (event driven processing)
- Electronic-Photonic conversion
- Hybrid integration

Precision of weights and variables

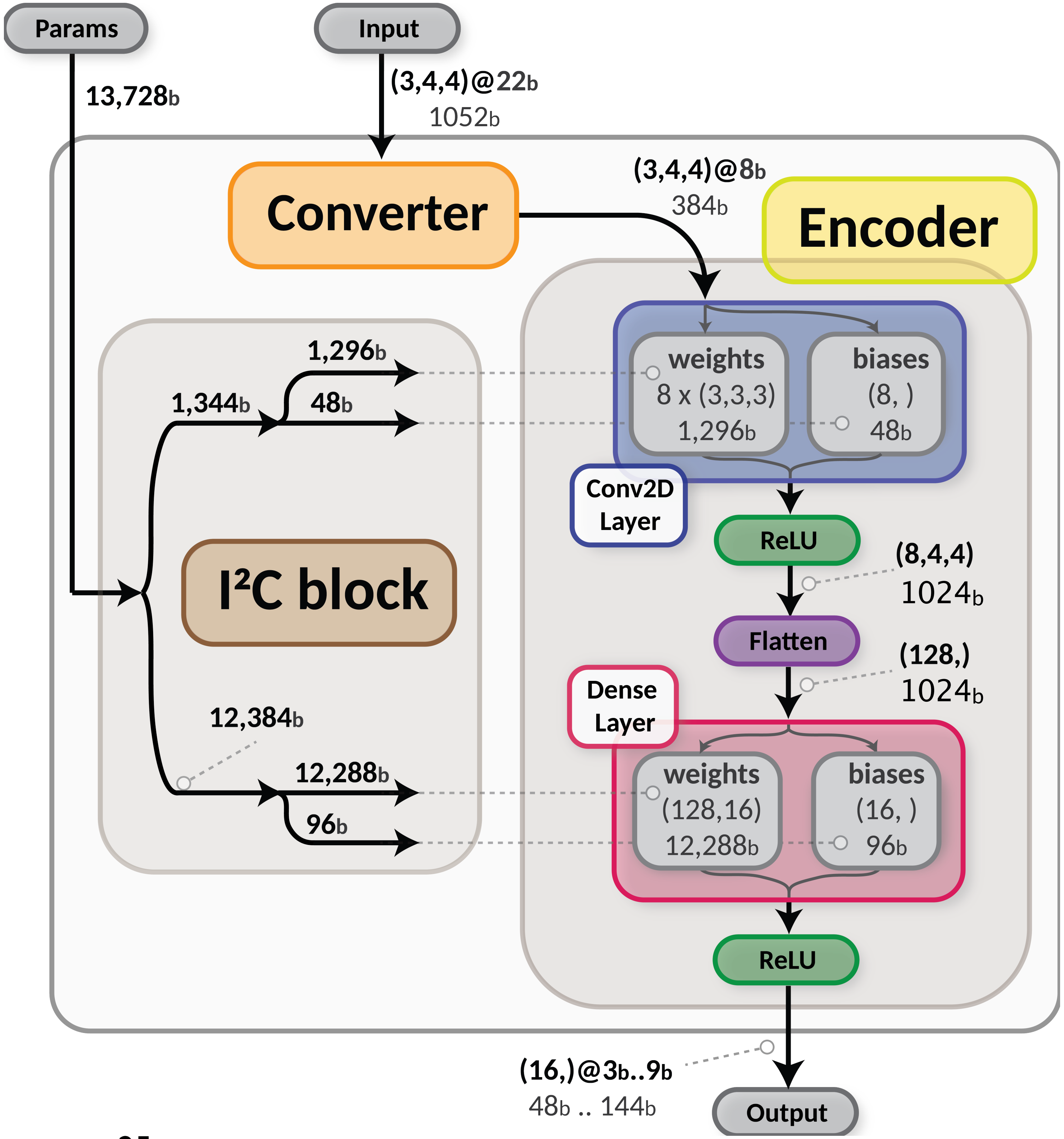
- Diagram is example for 4×4×3 reference network – same structure as final 8×8 network

- **Weights** are all 6b

For final 8×8 network:

- **hidden layer neurons:**
  - 8b fraction
  - sufficient integer bits to cover theoretical max value

- **output neurons**:
  - 9b total
  - 1b integer
  - covers maximum value from physics simulation


Params
Input
13,728b
(3,4,4)@22b
1052b
Converter
(3,4,4)@8b
384b
Encoder
1,296b
1,344b
48b
weights
8 x (3,3,3)
1,296b
biases
(8,)
48b
Conv2D Layer
ReLU
(8,4,4)
1024b
I²C block
Flatten
(128,)
1024b
Dense Layer
12,384b
12,288b
96b
weights
(128,16)
12,288b
biases
(16,)
96b
ReLU
(16,)@3b..9b
48b .. 144b
Output